# Solutions to Reinforcement Learning by Sutton Chapter 8

Yifan Wang

Aug 2019

***Exercise 8.1***

**In this particular problem, we have $\gamma = 0.95$ and $reward = 0$ except stepping into the goal state will give us $1$. If, we have a n-step method, we will have two conditions:**

**First, for states that cannot be reached backward from the goal state within n steps, or mathematically, $t + n < T$, we will have following updating formula ( or in similar forms )**

$$\begin{aligned} G_{t:t+n} &= 0 + \gamma^n Q_t(S_{t+n}, A_{t+n}) = 0.95^n\, Q_t(S_{t+n}, A_{t+n}) \\ Q_{t+n}(S_t, A_t) &\doteq Q_{t+n-1}(S_t, A_t) + \alpha\big[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\big] \\ &= Q_{t+n-1}(S_t, A_t) + 0.1\big[0.95^n\, Q_t(S_{t+n}, A_{t+n}) - Q_{t+n-1}(S_t, A_t)\big] \end{aligned}$$

**That brings a very difficult situation. Such update will be only meaningful if $Q_t(S_{t+n}, A_{t+n})$ is nonzero and accurate, (if we use control invariate one should not care about $Q_{t+n-1}(S_t, A_t)$). However, because initial Q are all zero, and because only states within the n-steps reach from the goal can be updated to nonzero, the update above is meaningless. Only $S_t$ for $t > T-n$ can be updated accurately. They will serve as basis for the second episode to make sure states $S'_t$ for $T - 2n < t' < T - n$. We can see the update is delayed. This is because n-steps method update states sequentially, but only the last parts of states are updated, so it needs time to transfer the updated information.**

**Even if n is large enough and we overlook the cost of computation**

**and memory, Dyna-Q is still better. This is why:**

**The planning part of Dyna-Q is very computationally efficient. We access an hash-table like data structure and we do only two multiplications. Besides its high efficiency, the loop itself is not limited within the n steps from the goal like in n-steps method. DynaQ randomly chooses S and A it experienced to exploit it. Despite the fact that such randomness will need an improvement, DynaQ successfully makes any pair $(S_t, A_t)$ to learn best of its Q value if such pair is indeed optimal or not. Because it is has a $\max_a Q(S', a)$ term, with random choice, if any Q value is changed during planning, it will be transferred to any adjacent $Q$s within the same state $S$ if planning choose them. By doing this, Dyna-Q eases out the time-irregularity of n-steps methods by random choice, decreases the computation cost by hash-table-sturcture and much less multiplications. This, is the huge advantages of Dyna-Q. Of course, it will have some exploration problem, that's addressed in Dyna-Q+.**

■

*Exercise 8.2*

**In first phase, in both examples, DynaQ+ finds the optimal policy much faster than DynaQ, that's why its slope rate quickly fixed to the optimal one and creates the gap when DynaQ was struggling. It is because DynaQ+ is so good at exploration to find the best policy. In second phase, both algorithms find the new optimal policy in the blocking example because both can handle the situation where environment gets worse. Again, we can see DynaQ+ is a faster. In shortcut example, DynaQ cannot find new policy because it is very hard for a vanilla $\epsilon$ method to keep exploratory across many steps. DynaQ+ are forced to explore further and creates the huge gap.**

■

*Exercise 8.3*

**This is because the gap is created by the faster convergence of DynaQ+ but the exploratory will have a cost. Such cost will force DynaQ+ to explore much more than DynaQ even it is already optimal. Such cost will gradually consumes the leading gap, that's why there is a narrowed sign. If the game continues without shortcut, DynaQ+ will have lower cumulative rewards than DynaQ but the optimal policy should remain the same.**

■

*Exercise 8.4*

**Here.**

■

*Exercise 8.5*

**In stochastic situation, model should maintain a possibility table of each reward it receives in any given pair $(S, A)$ that leads to $S'$. i.e, $P(r|S, S', A)$. It then sample the reward during planning. However, if the environment is changing, the task is more open and hard.**

**The first thought would be decay the appearing possibility of older rewards. Second would be decay the alpha when using old data. Third is using limited number of data and data of old rewards would be deleted. Beyond all this, one should always maintain a constant alpha because we cannot expect policy to convergence.**

**One attempt to solve the situation, (in my opinion), will be the following. First, we only accept limited number of newest reward data for each pair $(S, A)$ and a constant $\alpha$ to make sure we can learn from the newest. Then, maintain and track how sample means and sample variances of rewards change. If we observe from statistics methods**

**that the environment is stable, we decay the alpha, increase the limit of data size and accelerate the convergent. On the other hand, if stabilization cannot be detected, newest data capacity limit and constant alpha would be always hold.**

■

***Exercise 8.6***

**It will strengthen the sample updates. Consider an extreme case where $b = 10^{20}$ but only $10,000$ states occur during $99.99\%$ of time. The expected method would take forever to compute the true value. However, sample updates would get to almost 0 RMS error just like it did in the uniform case. Sample update performs better because, by its nature, it can ignore very unlikely states and emphasize the important, frequent ones without overlooking any one of them given enough computationally affordable number of samples.**

■

***Exercise 8.7***

**It is scalloped because it takes so much computation time to do a full sweep. Each update will bring a decent changes, especially in $b = 1$ case where the change of one state is fully transferred to other states. When $b$ is larger, the system is more stabilized due to the multiple sum of partial changes of following states, but also will need longer time to update and make the curve less scalloped. That is the reason why in 10,000 STATES case, the scallop is very dense and visible whereas in 1,000 STATES case, $b = 10$ case is the smoothest and $b = 1$ case has most visible scallop. In fact, if we use same scale on the computation time axis (x-axis), the scallop of lower part graph will be wider and harsher.**

■

*Exercise 8.8*

**PROGRAMMING.**

■

*Further Notice*

**This chapter is extremely interesting but lack of enough exercises. The learning shall not stopped by lack of them. And please allow me to give few exercises, with solutions later.**

***Additional Exercise 1:*** **In Example 8.4, page 170, Dyna-Q and Prioritized sweeping has different Gridworld size when it has huge increase of its updates. How to understand this phenomenon?**

■